System Modeling with Mixed Object and Data Models

Hessam Sarjoughian

Robert Flasher

Arizona Center for Integrative Modeling & Simulation

School of Computing and Informatics

IRA Fulton School of Engineering

{sarjoughian | robert.flasher}@asu.edu

Keywords: data engineering, model persistent, model transformation, system design, visual modeling.

Abstract

System designs are described from object and data centric perspectives. Component-based modeling approaches are well suited to specify structure and behavior of systems in terms of objects and relationships. Ontology-based modeling approaches describe a system's specification using data types and relationships. The Scalable Entity Structure Modeler is a component-based modeling framework suitable for specifying alternative system designs. This approach is extended with XML Schema to support unified dynamic objects and static data specifications for system design specifications. An example illustrating mixed component and data modeling is described using an extended realization of the SESM environment.

1. INTRODUCTION

Modeling is crucial for system analysis and design. An important problem in system modeling is how to create and manage multiple system architecture specifications. Enabling modelers to develop multiple models of a system based on sound principles, therefore, is highly desirable. To this end, various modeling languages and approaches have been developed to aid specifying hierarchical object and data model specifications at varying levels of abstractions.

Two of the main genres of modeling languages are targeted for developing software and simulation models. A common need is specifying alternative designs of a system. For example, a computer network model may specify a set of computers that are connected to one another using wireless connectivity. Such a model can be used to evaluate network topologies (e.g., communication range and its impact of quality of service) or to develop a design that can efficiently protect a wireless network against attacks. These models may be specified at varying levels of resolution. A computing node model may include probabilistic processing time or network traffic may be specified as primitive data or complex objects. Moreover, a complex system may have different architectures. For example, the wireless computer network may be modeled as a set of computers and switches having a hierarchical topology.

To support such needs, various modeling concepts approaches have been developed. Two of these modeling frameworks are DEVS [18] and UML [8]. They support modeling dynamics of simulation software models. Each is well-suited for a particular kind of modeling – i.e., DEVS is targeted for simulation modeling and UML for software modeling. A common theme among these approaches is to describe specific system models with strong emphasis on component. In contrast to these, XML [3, 14] is primarily used to model the static view of a system. It can also be used to describe static structures among simple and complex data elements. Other modeling approaches are Scalable Entity Structure Modeler (SESM) [2, 5, 7, 11], System Entity Structure (SES) [9, 16, 19], XML Schema, and DoD Architecture Framework (DoDAF). the Scalable Entity Structure Modeler (SESM) emphasizes a unified logical, visual, and persistent modeling framework for component-based model development [12]. The SES emphasizes modeling concrete alternative model structures. XML Schema allows describing arbitrary structures, but it does not have axioms that can establish similarity relationships among different structures of a system. DoDAF focuses on conceptual separation of models for describing complementary systems, operational, and technical views.

Given the above varying modeling approaches, it is desirable to support detailed object and data modeling. To achieve this goal, the importance of combined logical, visual, and persistent modeling is briefly described in Section 2. The SESM framework is detailed and the conceptual basis for its extension with XML Schema is

described in Section 3. In Section 4, the XML Schema models and its relationship with SESM are presented. In Section 5, a realization of the SESM environment is presented with an example model. A summary with a sketch of future work is given in Section 6.

2. BACKGROUND

Complex systems are commonly described by using a set of model abstractions and relationships among them. Modeling approaches such as Entity-Relation (ER) and Unified Modeling Language (UML) are used to model a system's specification from specific points of view. ER is used to describe a system's structure in terms of data entities and relations. The entities are generally simple, but can have intricate relationships which together may describe complex structures. UML describes objects and their relationships. UML classifiers represent various kinds of structures and behaviors. These classifiers may be combined together with data entities to describe complex dynamic systems. The ER and UML abstractions, therefore, support describing data-centric and object-centric structures, respectively. These modeling approaches are used to formalize different logical model abstractions.

The above modeling approaches offer capabilities to describe different kinds of logical models. Each of these approaches is grounded in a particular kind of modeling. Entity-Relation is targeted for describing structural, non-behavioral models. ER models lend themselves well for standardized logical model representation and model persistence in relational databases, but provide limited concepts and capabilities for visual modeling. UML is targeted for describing both object-oriented structure and behavior (models). The UML standard supports logical and visual modeling, but lacks a strong foundation for model persistence [6].

The System Entity Structure (SES) and XML modeling frameworks are aimed at ontological representation of high-level system and low-level data specification, respectively. SES is a labeled tree with a set of axioms that constrain the relationships among the tree's entities. Entities represent the parts of a system as a collection of similar, related structures.

As suggested above, models of a system can be given in terms of logical, visual, and persistent abstract model types. Each of these model types has its own syntax and semantics. The logical model type describes structure and behavior. The model specifications can be simple to complex – the ER models are generally low-level and do not describe behavior which makes them less complex as compared with the UML models. A logical model conforms to a common set of constructs and axioms. The syntax and semantics of the logical model type defines all model structures that consist of elements and relationships. The logical model may also define behavior as is in UML. The visual model type defines symbols for the logical models to support visual specifications of models. The model components and their relationships conform to the visual modeling language. The persistent model type specifies persistent memory patterns for the logical model structures across space and time. The model entities and relationships comply with syntax and semantics of databases.

The principal features of the SESM are scalable multi-aspect/resolution model specification, iterative/incremental model development process, and quantifying complexity metrics for models. The basic concept of the model types and their synthesis facilitate visual and persistent modeling. These capabilities afford automatic creation of well-formed model specifications according to the DTD and XML data language as well as object-oriented programming languages.

2.1. Data Engineering

An important application domain for SESM is data engineering which can be considered as system design with emphasis on data modeling. A major area of interest is handling of data sets obtained from Synthetic Aperture Radar (SAR) system. Modeling (or more specifically organizing) rich SAR data sets in a systematic way is essential for data gathering, processing, and analysis. For this purpose, the Universal Phase History Data (UPHD) standard has been developed using the SES and XML modeling approaches [17]. The SAR application domain is used to represent geospatial data obtained from sensors. Knowledge engineering of large data sets and complex relationships using the above modeling concepts is considered crucial for knowledge management, processing, and dissemination. This is because data sets need to be architected for data storage and making available processed data using technologies such as web-services.

Here, the data engineering concepts are used to extend the SESM modeling framework to support mixed object and data models. The extended approach described in the remainder of this paper supports separately modeling data and object models and their composition. The unified logical, visual, and persistent modeling framework supports developing data and object models that have rich structural and behavioral specifications. This in turn supports automatic transformations of SESM models to alternative specifications including DTD and XML Schema and semi-automatic transformation to simulation code.

3. SCALABLE ENTITY STRUCTURE MODELING FRAMEWORK

The Scalable Entity Structure Modeler is a modeling framework based on Entity-Relation (ER), System Entity Structure (SES), Object-Orientation (OO), visual modeling, simulation modeling, and model transformation (see Figure

1). The ER concepts support scaleable representation and storage of entities and their relations in databases. The concept for representing a system's structural representation is given by SES. The object-orientation composition, inheritance, and hierarchy are used for organizing alternative structures of a system. The visual modeling concepts offer visual abstractions that are key for systematic handling of tedious, error prone modeling tasks faced by designers and analysts. The simulation concepts are used to account for behavioral modeling of system specifications. Finally, well-formed exchangeable representations play a crucial role for generating alternative models that conform to standardized modeling languages such as XML and programming languages that can be executed with simulation engines.

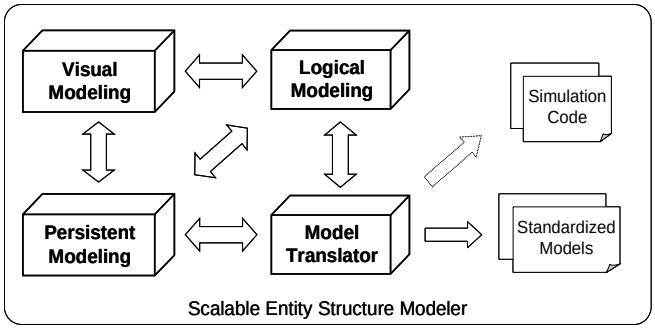


Figure 1: Logical, visual, and persistent model types with model translators

SESM is a component-based modeling approach in which families of system specifications are defined in terms of elementary Template, Instance Template, and Instance model types (refer to Figure 2) [10]. Each of the model types is defined to have primitive and composite model components. Every composite model component is a hierarchical tree where its leaf nodes must be primitive model components. A model component can be specialized such that its specializations are distinguishable. SESM defines a set of axioms that characterize compositional and specialization relationships across the elementary model types with support for object-oriented and markup languages (see Section 3.1). Two kinds of models are defined – i.e., simulatable components are used to define simple and complex dynamic models and non-simulatable models are used to define the static models (models that describe structure, but not behavior).

A realization of the SESM approach has been developed using Java and DBMS technologies [1, 5, 7]. SESM's underlying software architecture style is client/server. The first generation of SESM used the Oracle database [5] and subsequently was replaced with MS Access [1, 7]. A modeler can have multiple, independent modeling sessions, each with its own database. Next the specification of logical models is presented. The details of the persistence and visual models are deferred to [12].

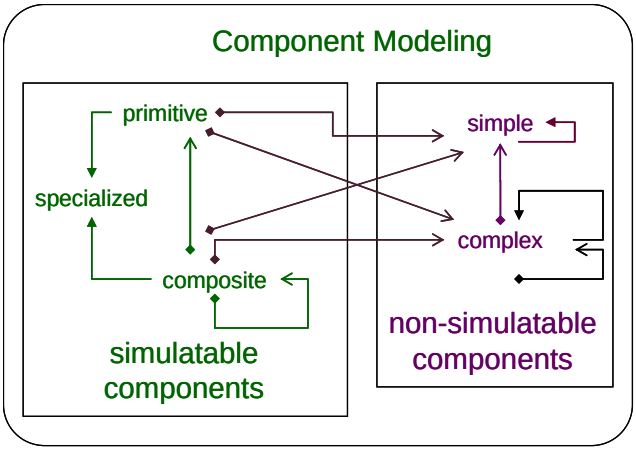


Figure 2: SESM component models

3.1. Logical Models

The primitive and composite model types are logical models. Each of these two model types has Template, Instance Template, and Instance models. A primitive Template Model can have a finite number of state variables. Each state variable is defined to have a type and may have an initial value. State variable types can be simple or complex objects including UML data types and classes. A primitive Template Model can also have inputs and outputs. Each input or output, which can be either simple or complex, is defined to have a unique port name. The collection of input and output ports for each component is defined as its interface. Input and output ports may be used to receive or send simple data or complex objects.

A primitive component can correspond to either a Template Model or an Instance Model. For the Template Model, its primitive component can be specialized using the is-a relationship. The term specializee is used to refer to the component that has a specialization relation to a component called specialized. The input/output interface of every specialized component is defined to be the same as the interface of its specializee. The state variables of specialized components may be different. A pair of specializee and its specialized can be distinguished based on their names. There are no specializee components for Instance Models. All primitive instance model components are distinguishable based on their assigned (or given) names.

A composite model corresponds to a Template Model, an Instance Template Model, or an Instance Model. A composite model consists of one to many primitive and/or composite components. The composite model and its

components have the same model type. A composite model may also have one or more states, inputs, outputs, and a set of links connecting the components that are contained within it. Any two components can send and receive information using links. Every composite component for a Template Model or Instance Template Model has a unique name and tree structure.

The allowed relationships among composite components are whole-part and is-a. Given a component, a sub-component and super-component composition relationship may exist only when no sub-component can be the same as its (immediate or higher) super-component. The sub-component is referred to as part and the component and super-component are referred to as whole. The number of components of a composite model can be either specified or left unspecified. A composite component can also be specialized as in a primitive model. Composite components can be used in multiple composite Instance Template Models. The hierarchy depth of a composite component is equal or greater than two. All instances of a composite component (corresponding to the Instance Model) are distinguishable from one another using their assigned (or given) names. The primitive and composite Instance Models are instances of their respective Instance Template Models. The whole-part and is-a relationships are constrained as described. The uniformity constraint – i.e., two components which have the same name have identical structures.

Instance models can only be generated from Instance Template models. An Instance model can be total or partial — i.e., a model hierarchy can be of any hierarchical depth depending on the model that is being transformed. For every model component that is specialized, one of its specializations must be selected to replace its specializee. If the number of sub-components of a component is left unspecified, the number needs to be determined when an Instance Template Model is transformed into an Instance Model.

The state variables and the input and output port variables may also be specified using data and object modeling languages. These simple and complex components and their composition and specialization relationships are defined with UML, DTD, or XML specifications.

3.2. Simulatable and Non-simulatable Models

To represent different possible structures of a system, it is important to use simple and complex non-simulatable model types. These model types referred to as non-simulatable models are distinguished from the simulatable Template, Instance Template, and Instance models. A simulatable model specification has a simulation protocol that dictates how the simulatable model is to be executed in (logical or real) time. In contrast, a non-simulatable models specified in UML may or may not have behavioral aspects. A model with behavior specification needs to provide its own execution regime with or without use of logical or real time.

A simulatable structure of a system has non-simulatable structures. In non-simulatable structures, it is important to specify the types for state or port variables. A state variable can have a complex type such as a list, and a port variable can have a simple type such as a string.

The semantics of the composition and specialization relationships used in UML are distinct from those that are defined for the Template, Instance Template, and Instance Models. The composition relationship among non-simulatable UML classes allows a class to have a composition relationship with itself. A class may have a dependency or realization relationship to another class. These relationships are not allowed in the Template Model. Similarly, the UML inheritance (i.e., specialization) relationship allows a child component to extend or restrict its parent component. In the Template Model, the specialization relationship is restricted to a specialized component to replace its specializee component. There are no extensions or restrictions between the specialized and specializee.

Given the distinct roles the simulatable and non-simulatable models play, the importance of differentiating them becomes evident. The abstractions defined for the Template, Instance Template, and Instance models are principally targeted for specifying alternative architectural system models, whereas the simple and complex models are intended for the specifications encapsulated within them. The simulatable model specifications can be transformed into simulation models that can be simulated. The non-simulatable model specifications can also be forward engineered into programming code. In the following section, the non-simulatable models are informally characterized in terms of the XML Schema language.

4. XML SCHEMA MODELS

Instead of using objects as basic ingredients for specifying dynamics of systems, it is useful to use data types as in XML Schema (XML-S). The data types, unlike objects, are void of behavior. Data types, in contrast to objects, offer a rich set of constructs to specify data elements, attributes, and data. The XML Schema language constructs allow describing structures having whole/part and is-a relationship. Also, unlike object-based modeling languages the language supports creating and using simple and complex data types.. For example, an element can contain text and unconstrained child elements or contain text with strict rules.

Conceptually XML-S primitive and complex data types are similar to the SESM primitive and composite model

component (see Figure 3). The SESM primitive and composite model components can be used to specify XML-S primitive and composite data types. The state variables of the SESM model components can also be specified as XML-S primitive and composite data types. The state variables can be specified using object-orientation and XML-S modeling languages according to the degree of sophistication required. For example, an element can have simple content – a name having simple data type String. The element containing text conforms to a specific data type such as Integer. Alternatively, a `simpleContent` element can have attributes with constrained content. The content is simple data types and can hold attributes with extension or restriction elements. While these kinds of specifications are possible to specify using the extensibility afforded by UML MOF or others (e.g., by extending the SESM modeling approach), the XML-S standard is well suited for such data modeling.

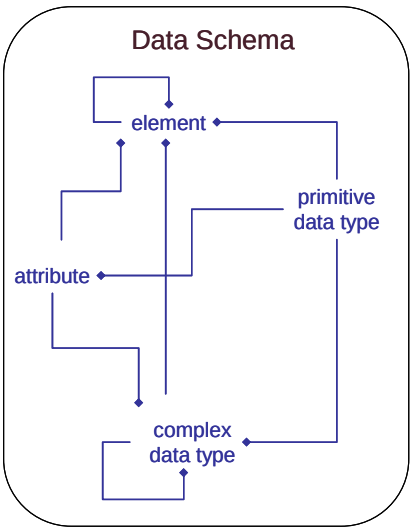


Figure 3: SESM XML Schema data models

The XML-S sequence and choice elements serve a role similar to the composition and specialization relationships defined for SESM. The sequence element defines the enclosed elements that appear both in the instance structure and declaration. The choice element defines required and exclusive elements that are enclosed in an element. These elements can be combined to specify complex models as supported in SESM using the whole/part and is-a relationships defined in SESM.

As shown in Figures 2 and 3, on the one hand, the SESM simulatable components can be used to represent XML-S primitive and complex data types. On the other hand, UML and XML-S languages can be used to represent SESM non-simulatable components. A key benefit is the SESM foundational concept of a unified logical, visual, and persistent modeling with capability to model object structures and behaviors as well as complex structured data models enabled by XML-S.

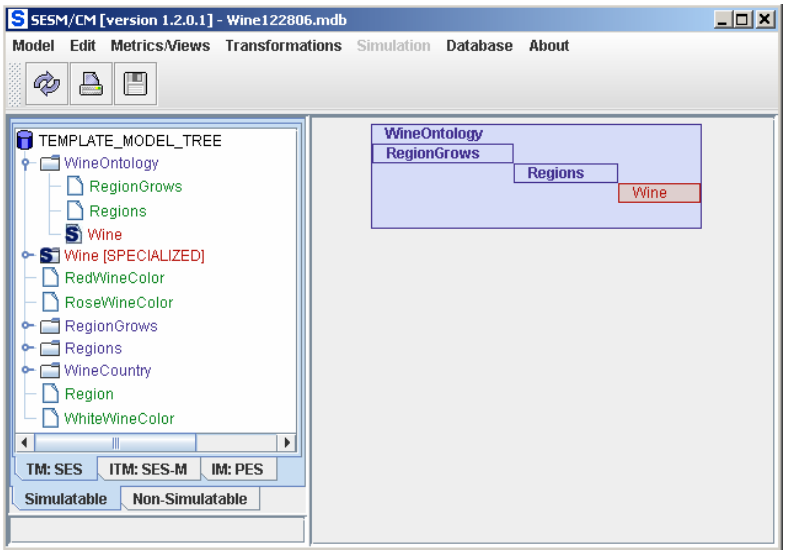
The other capabilities are forward and reverse engineering. With forward modeling, XSD and DTD models can be derived from SESM specifications. With backward modeling, XSD and DTD models that are consistent with SESM can be put into the database and thus support visual modeling, analysis of model complexity (i.e., using complexity metrics). These two capabilities offer a round-trip modeling approach for specifying alternative system designs. Furthermore, component-based simulation code can be generated semi-automatically. In particular, DEVSJAVA atomic and coupled models can be generated from SESM models. The SESM environment supports partial specification of atomic models (i.e., input/output ports, state variables, and the skeletons of the transition functions). Coupled models can be specified completely (i.e., input/output ports, couplings, and hierarchical decomposition).

5. SESM ENVIRONMENT

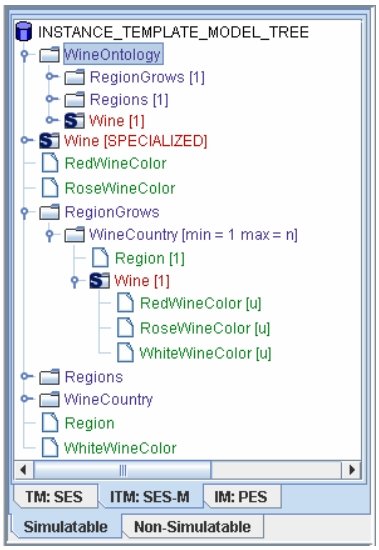
Before presenting an example model, an overview of the SESM environment is given. This environment consists of three parts: Client, Server, and Network. All write operations requested by a client are managed through the Network part and processed by Server. The Server enforces the axioms of SESM and consequently legitimate client operations are handled by the database. All read operations are directly handled by the database.

As shown in Figure 4, a client can model a system in terms of the simulatable and non-simulatable modeling elements. In the left-hand frame, there are two main tabs: Simulatable and Non-Simulatable. The Simulatable tab has three tabs corresponding to the Template, Instance Template, and Instance models. The menu item *Model* shown in Figure 4 allows a user to create Template, Instance, and Data Type models. The tree structures of the Template, Instance Template, and Instance models are shown in each tab and can be manipulated (i.e., new models can be added, modified, and deleted). The Non-Simulatable tab has two tabs corresponding to the non-simulatable models (NSM) and Data Types. The simulatable and Data Types models are stored in the database.

In the right panel, a client can view the color coded block models of the TM, ITM, and IM models. Rounded rectangles are used to visually represent primitive components. A rectangle is used to identify composite and specializee components. Components with a multiplicity range are shown as rectangles with dashed lines. The same color coding is used to differentiate model types in the model tree representations.



(a): Template tree and block models



(b): Instance template model

Figure 4: SESM UI for tree and block model specification

The tree representation uses 'folder' and 'page' visual notations with a letter S to distinguish specializee models. The SESM naming convention tabs for the Template, Instance Template, and Instance models include the SES terms to aid modelers working with the SES concepts. The SES trees can be represented with TM and IM models and the pruned entity structure (PES) can be represented with IM. The block model components are placed diagonally for more efficient and simple visual representation and manipulation. The ordering of the block models (which do not have ports and couplings) for the `WineOntology` composite model shown in Figure 4(a) does not have any specific semantics within the SESM modeling framework. The order of `RegionGrows`, `Regions`, and `Wine` is based on the order in which they are added to the `WineOntology` model.

Visual modeling of coupling relationships, specification of states (variables and types) and ports (port names, variables, and types) are supported in this panel. Complexity metrics and translation to XML and simulation code are supported in the tree structures, block models, and the menu items (*Edit*, *Metrics/Views*, and *Transformations* lists). The *Model* menu item supports creating Template Model, Instance Model, and Data Type elements. These Data Types such as the one shown in Figure 5 complement the NSM models [2, 11] (see Section 3.2).

The *Database* menu item which is available in the main menu can be used to initialize the model (i.e., removing all entries in the database – simulatable and Data Type models). Changes to NSMs are not supported within SESM; instead creation, modification, and deletion can be performed using other means (e.g., UML tools, Eclipse Modeling Framework [4], and XML-Spy [15]). The structural complexity of every template model is also available (see Figure 6) [7, 11].

```
Data Type Summary

Name:  dewPoint
Subtype:    simpleContent
Subtype Definition:
    <xs:simpleContent>
        <xs:extension base="xs:int">
            <xs:attribute name="level" type="xs:string" />
        </xs:extension>
    </xs:simpleContent>
```

Figure 5: XML-S specification for `dewPoint` model

Attribute	Name	Type	Value
Model	Region	ATOMIC	
Input variables			
Output variables			
State variables			
	location	geoPosition	West
	aveDewPoint	dewPoint	moderate
NSM variables			

Figure 6: Behavioral metrics for `Region` model

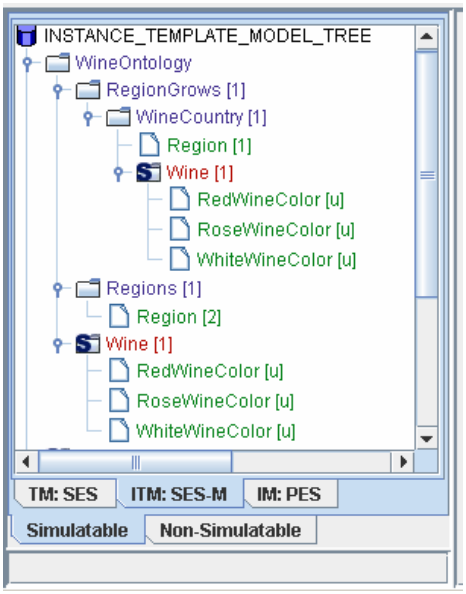


Figure 7: Multiplicity for the `WineCountry` model

Given the combined SESM and XML-S specification, every Instance Template model can be transformed into an XSD specification. Given the expressiveness of SESM, it can be used to visually specify persistent SES models. Given the Instance Template models such as those for the `WineOntology`, Instance models can be generated from them. If an Instance Template model has a multiplicity range, the user first chooses a desired multiplicity. For example, `WineCountry` has multiplicity ranging from 1 to n (see Figure 4(b)). Figure 7 shows the multiplicity for the `WineCountry` model to be 1.

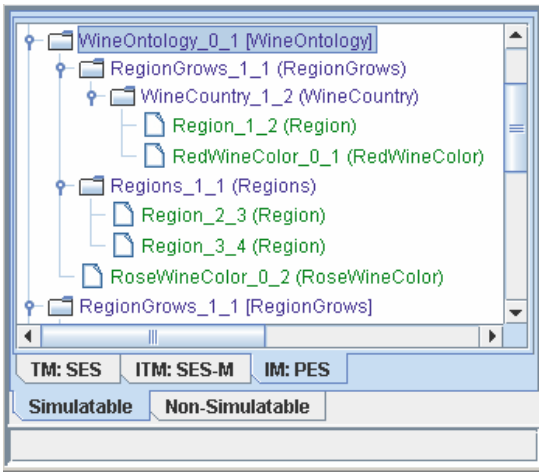


Figure 8: `WineOntology` Instance model

Once all multiplicities are determined, instance models can be generated for every primitive and composite Instance Template model. The Instance model for the `WineOntology` model is shown in Figure 8. This model has `RegionGrows_1_1`, `Regions_1_1`, and `RoseWineColor_0_2` which is a specialized from the `WineColor` primitive Template model. Similarly, `Region_1_2` and `RedWineColor_0_1` instance models are generated for the `WineCountry_1_2` instance model. Finally, `Region_2_3` and `Region_3_4` instance models are generated for the `Regions_1_1` instance model.

The element Region, shown in Figure 9, is a well-formed XML-S model transformed from the SESM primitive component `Region` along with XML-S `dewPoint` and `GeoPosition` data types. Such XML-S or DTD models can be automatically generated for every model that is developed using DTD and XML-S translators added to the SESM environment. The generation of an instance model in SESM results in a model that can have the same representation as those that can be generated with the SES/DTD and SES/XML translators developed for SES-Java [13]. This requires (i) there is a correct mapping between SES and SESM and (ii) the representation of XML or DTD data types is also supported in the same way for SES and SESM [12]. For example, the resulting XML-S shown in Figure 9 represents a leaf entity of an SES model that has attributes specified according to the XML-S data types.

```
- <xs:simpleType name="geoPosition">
  - <xs:restriction base="xs:String">
      <xs:enumeration value="East" />
      <xs:enumeration value="North" />
      <xs:enumeration value="West" />
      <xs:enumeration value="South" />
    </xs:restriction>
  </xs:simpleType>
- <xs:complexType name="dewPoint">
  - <xs:simpleContent>
    - <xs:extension base="xs:int">
        <xs:attribute name="level" type="xs:string" />
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
- <xs:element name="Region">
  - <xs:complexType>
    - <xs:attribute name="location" type="geoPosition" use="required">
        <xs:enumeration value="West" />
      </xs:attribute>
    - <xs:attribute name="aveDewPoint" type="dewPoint" use="optional">
        <xs:enumeration value="52" />
      </xs:attribute>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Figure 9: XML-S `Region` model

6. CONCLUSIONS

The importance of modeling complex systems from complementary object and data perspectives were described. This offered a motivation for extending the SESM modeling framework with XML. The simulatable and non-simulatable model types, roles, and relationships were presented. The collective capabilities of object and data model development were demonstrated using the extended SESM realization. It offers a step forward toward realization of multifaceted model development as proposed with DoDAF. The SESM framework supports system design from object-centric and data-centric perspectives. With the addition of the XML models, design specifications can be supported from simulation and non-simulatable perspectives. The use of the SESM modeling framework for specifying the class SES/XML models was described. Future work includes application of the SESM framework for simulation-based design and automated testing. The SESM environment may be used toward engineering of data intensive systems with capability to process data available across grid networks and delivery of information using web-services.

Acknowledgement

This research is supported in parts by grants from NSF, Intel Corporation, and Northrop Grumman. We would like to acknowledge the fruitful discussions regarding the Data Engineering project with Bernard Zeigler of the University of Arizona and Philip Hammonds, Rodney Leist, Steven Madden, and Chad Schulenberg of JITC/DISA and Northrop Grumman.

References

1. Bendre, S., Behavioral Model Specification Towards Simulation Validation Using Relational Databases, in Computer Science and Engineering, 2004, Arizona State University: Tempe, AZ, p. 1-150.
2. Bendre, S. and Sarjoughian, H.S. Discrete-Event Behavioral Modeling in SESM: Software Design and Implementation, Advanced Simulation Technology Symposium, 2005. p. 23-28, San Diego, CA.
3. Bradley, N., The XML Schema Companion, 2004: Addison Wesley.
4. Budinsky, F., Steinberg, G., Merks, E., Ellersic, R., and Grose, T., Eclipse Modeling Framework, 2003: Addison-Wesley.
5. Fu, T.-S., Hierarchical Modeling of Large-Scale Systems Using Relational Databases, in Electrical and Computer Engineering, 2002, University of Arizona: Tucson, AZ, p. 1-114.
6. Jordan, D. and Russell, C., Java Data Objects, 2003: O'Reilly.
7. Mohan, S., Measuring Structural Complexities of Modular, Hierarchical Large-scale Models, in Computer Science and Engineering, 2003, Arizona State University: Tempe, AZ, p. 1-112.
8. OMG. Unified Modeling Language, 2004, http://www.omg.org/technology/documents/formal/uml.htm
9. Park, H.C., Lee, W.B., and Kim, T.G., RASES: A Database Supported Framework for Structured Model Base Management, Simulation Practice and Theory, 1997, 5(4), p. 289-313.
10. Sarjoughian, H.S., An Approach for Scaleable Model Representation and Management, 2001, Computer Science & Engr., Arizona State University: Tempe, AZ, p. 1-9.
11. Sarjoughian, H.S. A Scaleable Component-based Modeling Environment Supporting Model Validation, Interservice/Industry Training, Simulation, and Education Conference, 2005. p. 1-11 Orlando, FL.
12. Sarjoughian, H.S., Fu, A., Bendre, S., and Flasher, R., A Unified Logical, Visual, and Persistent Modeling Framework, in-preparation.
13. SES-Java. Virtual Work Table, http://www.devsworld.org/, 2006.
14. XML. eXtensible Markup Language, 2005, http://www.w3.org/XML/.
15. XMLSpy. XML editor for modeling, editing, transforming, & debugging XML technologies, 2006.
16. Zeigler, B.P., Multi-Facetted Modeling and Discrete Event Simulation, 1984, New York: Academic Press.
17. Zeigler, B.P. and Hammonds, P.E., Modeling&Simulation-Based Data Engineering: Introducing Pragmatics into Ontologies for Net-Centric Information Exchange, in-press, 2006.
18. Zeigler, B.P., Praehofer, H., and Kim, T.G., Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems, Second Edition, 2000: Academic Press.
19. Zeigler, B.P. and Zhang, G., The System Entity Structure: Knowledge Representation for Simulation Modeling and Design, in Artificial Intelligence, Simulation and Modeling, K.A.L. L.A. Widman, and N. Nielsen, Editor. 1989, John Wiley. p. 47-73.